
# PYRAMID User's Guide

**John Z. Lou, Charles D. Norton, & Thomas A. Cwik**

NASA Jet Propulsion Laboratory
California Institute of Technology
MS 168-522
4800 Oak Grove Drive, Pasadena, CA, 91109-8099, USA
http://www-hpc.jpl.nasa.gov/APPS/AMR

| | |
|---|---|
| Version: | 1.0 |
| Platforms: | Cray T3E and Scalar Workstations |
| Requirements: | Fortran 90, C, MPI |

Contact: {John.Lou, Charles.Norton, Thomas.Cwik}@jpl.nasa.gov

Adaptive mesh refinement (AMR) is an advanced numerical technique gaining popularity in the scientific and engineering computing community for solving large-scale computing problems. The use of AMR techniques can significantly improve computational efficiency, and computer memory efficiency, by devoting finite computing resources (CPU time, memory) to the computational regions where they are most needed, making it possible to compute an accurate numerical solution with much less computing resources compared to the use of a global fine mesh. AMR algorithms and software development for uniprocessor computers have been investigated in the computational fluid dynamics (CFD) community for many years, and applied successfully to various CFD problems as reported in the open literature. Development of parallel adaptive mesh refinement algorithms and software tools, however, is still a relatively new research area. Some work in the development of parallel AMR components have been reported for structured and unstructured meshes in recent years.

Our goal is to develop a comprehensive parallel AMR Library which can be easily integrated into existing mesh-based parallel applications running on massively parallel computers and computer clusters. The software platform for our parallel library is based mainly on Fortran 90 and the Message-Passing Interface (MPI) standard, a choice which we think offers a reasonable combination of flexibility, safety, (compared to C and C++), portability, and efficiency. Interfaces of the parallel AMR Library to other popular scientific computing programming languages like Fortran 77 and C will be provided. The parallel Library will include a set of components for operations at various AMR stages, which include a parallel mesh partitioner, a parallel adaptive mesh refiner with quality control, a parallel load-balancing module, and a parallel local-error estimator. Our software architecture design for the parallel AMR library will make these AMR components highly modular, maximizing the user's control of individual components, providing the simplest possible interfaces among the components and the application code.

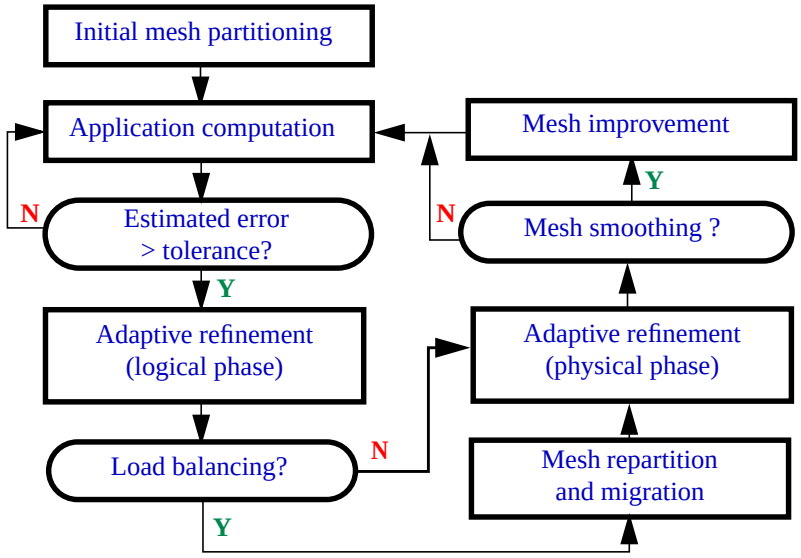
# 1.0 PYRAMID Software Architecture

Parallel unstructured adaptive mesh refinement (AMR) is an advanced numerical technique useful for time-dependent problems over a non-uniform structural domain. When the application region is discretized, various portions of the computational domain can be refined, or coarsened, in regions where additional accuracy is required (based on error-estimation). This approach saves memory and time over methods that simply refine the entire domain, but for sufficiently large problems parallel computers are required.

## 1.1 Organization Overview

The general organization of the parallel AMR process is illustrated. Initially, the (generally random) input mesh must be repartitioned and redistributed after loading from the disk. The application computation and local error-estimation step occur, followed by the logical AMR process. The logical AMR stage determines how elements will be refined, but the physical refinement of elements is delayed. Load balancing can occur based on this process.

**FIGURE 1.** Illustration of Parallel AMR Procedure.




The load balancing process moves coarse elements, based on a weighting scheme, to the proper destination processors using the migration module. (We use the ParMeTiS mesh partitioner from University of Minnesota to assist in this stage.) At this point, the physical AMR step occurs by a local refinement process based on the logical refinement stage.

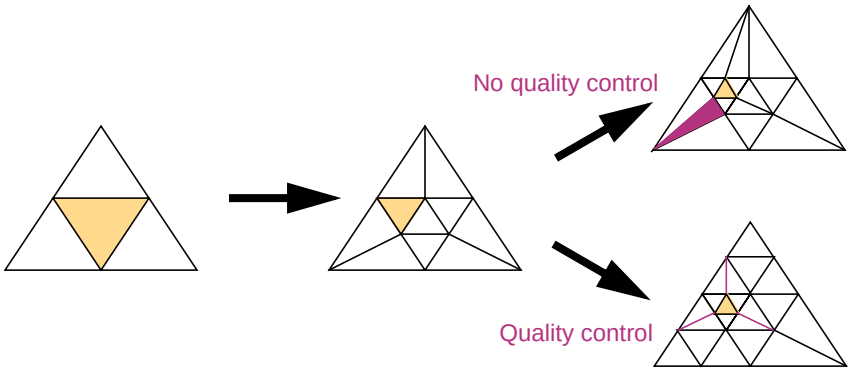
Finally, the element quality can be checked either by performing an explicit mesh smoothing operation or by ensuring high quality element creation during refinement. We perform the latter since mesh quality control can be introduced as part of the logical refinement stage without the potential complications of explicit mesh smoothing applied after physical refinement of the mesh.

## 1.2 Load Balancing, Mesh Migration, and Quality Control

The ParMeTiS library tries to minimize the edge cut length, and data movement, among the processors by examining a dual-graph representation of the mesh. Since ParMeTiS only returns a partitioning, the migration module constructs a dual-graph representation of the mesh data structure, analyzes the load balanced partitioning returned from ParMeTiS, and migrates mesh components among processors automatically, all in parallel.

The mesh quality control scheme classifies elements based on how they were refined. This allows us to foresee the potential of creating elements with poor aspect ratios. When identified, we can replace these elements with a refinement pattern that improves upon the geometry.

**FIGURE 2.** Illustration of the Quality Control Process




The figure shows the original refinement of a coarse element. Successive refinements will destroy the aspect ratio of existing elements, leading to poor mesh quality. The approach we apply modifies the coarse element refinement, as shown, should either of the child elements require further refinement (due to local errors or mesh consistency constraints from neighbor element refinement). This process controls the mesh quality, at the expense of creating slightly more elements.

## 1.3 Working with PYRAMID Library Objects

Organizing and programming the data structures for parallel AMR is very complicated. The main structure is the computational mesh that represents a complex geometry with many components. Object-Oriented methods, using C++, have been applied to manage the complexity for this problem before since AMR components can be designed, and

relations can be specified, using this paradigm. Although parallel AMR is growing in importance, it has not been widely accepted for a number of reasons. These include the programming complexity and the lack of data abstraction techniques available to the experienced Fortran 77 programmer.

Fortran 90, however, contains many new features that provide new alternatives for parallel AMR. These features allow for the code to be designed using exactly the same abstractions that exist in alternative language implementations, but in a Fortran framework more familiar to most computational scientists.

Our basic philosophy has been to apply object-oriented design principles in AMR data structure organization. Such features allow for a very abstract design and representation of source code for parallel AMR. A main program that loads a mesh, distributes it among the parallel processors, creates the mesh data structure, performs repartitioning and visualization, now looks like this:

**FIGURE 3.** Illustration of Adaptive Refinement Procedure with Fortran 90 Library Objects

```
program pamr
use pyramid_module
      implicit none
     ! statements omitted
      type(mesh) :: in_mesh
        call PAMR_INIT()
        call PAMR_CREATE_INCORE(in_mesh, “mesh_data”)
        call PAMR_REPARTITION(in_mesh)
        do i = 1, refine_level
            call PAMR_ERROR_EST(in_mesh)
            call PAMR_LOGICAL_AMR(in_mesh)
            call PAMR_REPARTITION(in_mesh)
            call PAMR_PHYSICAL_AMR(in_mesh)
        end do
        call PAMR_VISUALIZE(in_mesh, "visfile.plt")
        call PAMR_FINALIZE(.true.)
end program pamr
```

The main object that a user manipulates is the mesh object via library routines. Many PYRAMID library calls accept optional arguments providing functionality beyond the basics shown above.

While Fortran 90 is not an object-oriented language (certain OO features can be emulated by software constructs) the methodology simplifies library interfaces such that the internal details are hidden from library users. This is an important goal of our design.

## 2.0 Performance Characteristics (CrayT3E)

These performance results show the scalability of the code in terms of the adaptive refinement time and the load balancing time. The element number varies since, in this illustrative example, refinement randomly chooses half of the elements per partition where mesh consistency must be maintained. The AMR Time includes the time to perform logical and physical refinement while the load balancing time includes partitioning, mesh redistribution, and data structure reorganization. These are CPU clock measurements.

**TABLE 1.** Performance Results for Waveguide Filter after 3 Refinements

| Number of Processors | AMR Time (seconds) | Load Balancing (Migration) Time in sec | Number of Elements | Elements per Processor |
|---|---|---|---|---|
| 32 | 57.34 | 15.36 | 292,612 | ~9,144 |
| 64 | 13.55 | 3.75 | 295,405 | ~4,615 |
| 128 | 2.93 | 1.65 | 305,221 | ~2,384 |
| 256 | 0.54 | 1.51 | 335,527 | ~1,310 |
| 512 | 0.27 | 1.86 | 397,145 | ~775 |

## 3.0 Installation and Usage Instructions

Our library only handles the adaptive refinement features associated with a finite element process. We expect that library users provide an input mesh and an indication of the error on each element, meeting library specifications. The library will perform an adaptive procedure and return modified versions of the input data. It is very easy to install the library on a workstation or Cray T3E system.

### 3.1 Building the PYRAMID library

1. Unpack the source code with "**gunzip pyramid.tar.gz**" and "**tar xvf pyramid.tar**".
2. Examine the *README* file and set makefile parameters for a parallel Cray T3E installation or a scalar workstation installation (Cray T3E is the default).

   Note that "**SEQ/mpif.h**" *must* be in the Pyramid.v1.0/ directory for a scalar installation while it *must not* be in that directory for a parallel installation.
3. Build the library with "**make ppyramid**" or "**make spyramid**" for a parallel or scalar workstation installation respectively. Typing "**make**" will build the parallel version by default.

4. Compile the demo program, using the *Makefile.pamr* file, by typing "**make pamr**" or "**make samr**" for the parallel or scalar version respectively.

   Note that the input mesh file for the demo is called "**mesh_data**".

5. Run the demo program with "**mpprun -n <procs> pamr_test**" for the parallel version or "**pamr_test**" for the scalar version. (<procs> = 8 is sufficient for testing.)

6. A visualization file called *pamr_mesh.plt* will be created. This can be viewed with TecPlot using "**tecplot pamr_mesh.plt**". Note that you may need to adjust TecPlot's mesh and shade attributes to view the partitioning correctly.

   In TecPlot, under the *Field* option select *Mesh Attributes*. Move to *Zone Num* and select all zones, then under *Mesh Color* select black. Return to the *Field* option and select *Shade Attributes*. Under *Shade Color* make sure that every zone has a unique color. Make sure that the *Mesh*, *Shade*, and *Boundary* buttons are selected from the main control box and select *Redraw* to show the mesh. Colors represent separate partitionings.

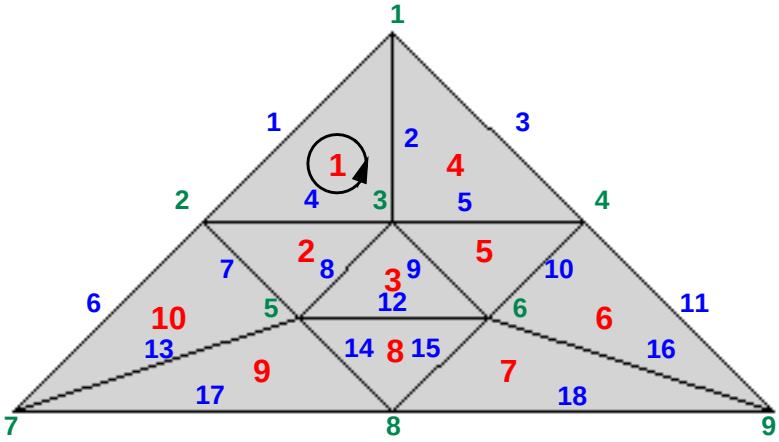
The sample user program Makefile.pamr shows how the libpyramid.a and ParMetis libraries are linked into the user code. Note that -M<libpath> , -p <libpath>, or a similar compiler option is also required make the module libraries visible.

## 3.2 Mesh Data Format

The mesh data format is illustrated below. One key point is that the mesh data must follow the counter-clockwise convention. That is, the direction in which edges are listed is related to the order in which nodes are listed using a counter-clockwise convention. This is generally a well accepted standard in generating mesh data files.

In the figures below, we show a sample mesh, its labeling, and the properly formatted input file for the mesh. The counter-clockwise orientation for element 1 is shown.

**FIGURE 4.** Illustration of a sample input mesh



By examining the mesh labeling above, and the input data format below, the counter-clockwise format will become clear. Here are additional points to remember when creating the input data file from the mesh description.

1. The input mesh file must be named "**mesh_data**".
2. The first line lists **#nodes, #elements, #edges,** and an **arbitrary** value, in that order. The AMR library will verify these values when the mesh is loaded.
3. The **node coordinates** are listed next.
4. The **node numbering** follows the node coordinates.
5. The **edge numbering** follows the node numbering
6. Finally, the **edge node endpoints** follow the node numbering.

Note that when listing node coordinates that the node number comes first, followed by the (x,y) coordinate pair. When listing mesh nodes the element number and a dummy value are followed by the counter-clockwise ordering of nodes for that element. The mesh edges list the mesh element number followed by the edge ordering in a counter-clockwise fashion based on the node numbering. The edge node endpoints list the edge number followed by the node endpoints and a dummy value. This is used to verify and/or correct the counterclockwise ordering of previous mesh components.

In the figure above, the elements, nodes, and edges are numbered according to the sample input mesh format in the figure below. The structure of this format must be followed exactly.

**FIGURE 5.** Illustration of Input Mesh Format

```
9 10 18 1

1
8. 8.
2
4. 4.
3
8. 4.
4
12. 4.
5
6. 2.
6
10. 2.
7
0. 0.
8
8. 0.
9
16. 0.
```

1, 1
1 2 3
2, 1
3 2 5
3, 1
3 5 6
4, 1
1 3 4
5, 1
4 3 6
6, 1
4 6 9
7, 1
6 8 9
8, 1
6 5 8
9, 1
5 7 8
10, 1
5 2 7

1
1 4 2
2
4 7 8
3
8 12 9
4
2 5 3
5
5 9 10
6
10 16 11
7
15 18 16
8
12 14 15
9
13 17 14
10
7 6 13

1
1 2 0
2
1 3 0
3
1 4 0
4
2 3 0

```
5
3 4 0
6
2 7 0
7
2 5 0
8
3 5 0
9
3 6 0
10
4 6 0
11
4 9 0
12
5 6 0
13
5 7 0
14
5 8 0
15
6 8 0
16
6 9 0
17
7 8 0
18
8 9 0
```

### 3.3 Error Handling

The library performs various status checks during operation, especially regarding memory usage. If a library routine fails due to insufficient memory this is considered a fatal error and the library will abort immediately.

If a library routine fails due to an internal PYRAMID message this indicates that an abnormal, or unexpected, condition has occurred. The library will abort, but such messages should be brought to the attention of the authors.

### 3.4 Parallel and Serial Usage

A user source code can use the library on a parallel machine or a scalar workstation without change. Note, however, that the build procedure is different.

## 4.0 Library Commands

The library commands are very easy to use. The features and options available are described. Note that PAMR_INIT() and PAMR_FINAL() must be called at the begin-

ning and the end of the parallel AMR process, exactly once. No check is made to ensure that any routine is called in a particular order.

## 4.1 Initialization Commands

### PAMR_INIT()

This routine initializes the PYRAMID library, and MPI if necessary.

### PAMR_CREATE_INCORE( this, in_mesh_file )

```
type (mesh), intent (inout) :: this
character (len=*), intent (in) :: in_mesh_file
```

This routine reads non-distributed mesh components (a single file in JPL format) into a single processor and distributes them across multiple processors.

### PAMR_CREATE_OUTOFCORE( this, in_mesh_file )

```
type (mesh), intent (inout) :: this
character (len=*), intent (in) :: in_mesh_file
```

This routine reads distributed mesh components (multiple files in JPL format) into multiple processors. An auxiliary Fortran 90 program "PAMR_MESH_FORMAT" can create the distributed mesh components in the proper file format on a workstation. This format allows for the initial loading of very large meshes.

### PAMR_CREATE_COMP( this, in_edges, in_nodes, in_node_coords )

```
type (mesh), intent (inout) :: this
integer, dimension(:,:), intent (in) :: in_edges
integer, dimension(:,:), intent (in) :: in_nodes
real, dimension(:,:), intent (in) :: in_node_coords
```

This routine reads distributed mesh components into multiple processors from data stored in Fortran 90-style two-dimensional arrays. (Also valid for Fortran 77.)

### PAMR_CREATE_COMP( this, in_edges, in_nodes, in_node_coords )

```
type (mesh), intent (inout) :: this
integer, dimension(:), intent (in) :: in_edges
integer, dimension(:), intent (in) :: in_nodes
real, dimension(:), intent (in) :: in_node_coords
```

This routine reads distributed mesh components into multiple processors from data stored in Fortran 90-style one-dimensional arrays. (Also valid for Fortran 77.)

## 4.2 Termination Command

### PAMR_FINALIZE( mpi_finalize )

```
logical, intent (in), optional :: mpi_finalize
```

This routine terminates the PYRAMID Library. If mpi_finalize is present and true, it will also terminate MPI.

### 4.3 Adaptive Refinement Commands

**PAMR_LOGICAL_AMR( this )**
type (mesh), intent (inout) :: this

Compute mesh refinement pattern on a coarse mesh without actually performing the physical refinement.

**PAMR_PHYSICAL_AMR( this )**
type (mesh), intent (inout) :: this

Perform the physical mesh refinement based on the logical AMR process.

**PAMR_ERROR_EST( this )**
type (mesh), intent (inout) :: this

Apply the error estimation process on mesh elements. Note: Error estimation is not strictly a part of the library. Users can provide a list of error estimates on an element-by-element basis.

### 4.4 Repartitioning and Data Migration

**PAMR_REPARTITION( this )**
type (mesh), intent (inout) :: this

This routine interfaces to the ParMeTiS mesh partitioner, written in C, and migrates the reorganized mesh components to the proper processors.

### 4.5 Visualization

**PAMR_VISUALIZE( this, vis_file )**
type (mesh), intent (inout) :: this
character (len=*), intent (in), optional :: vis_file

This routine generates a visualization file, in TecPlot format, from the distributed mesh data structure. The file name is “pamr_mesh.plt” unless the optional argument is specified.

## 5.0 Closing Comments and Future Work

The PYRAMID library is still in development, but this early release allows us to receive user feedback regarding the system. Please contact the authors with any comments or suggestions that would improve your use of the library. Updates to the library will occur

frequently, as it becomes more wide spread. Visit the web site, or ask to be placed on our mailing list to be notified of changes and enhancements.

A system to support 3D parallel adaptive mesh refinement is nearing completion. Watch the web site for more information on this system.

Future versions of the library will support mesh coarsening, additional mesh input formats, and more visualization capabilities.